فاز اول پروژه اصول طراحی کامپایلر

در این پروژه قصد داریم یک سری دستورات ساده و مرکب برای یک زبان برنامه نویسی فرضی را طراحی کنیم و کامپایلر آن را با استفاده از زیرساخت LLVM بسازیم. در نهایت object code آن را با کمک LLVM نمایش می دهیم.

قبل از آن که به شرح دستورات تعریف شده در زبان بپردازیم، عبارت های ریاضی و منطقی مجاز در این زبان برنامه نویسی را شرح می دهیم:

به کمک عملگر های محاسباتی + ، - ، * ، / ، % ، ^ ، ( ) می توان عبارت های ریاضی متنوعی ساخت. به کمک عملگرهای رابطه ای < ، > ، <= ، => ، == ، =! و عملگرهای منطقی xor , or , and و با استفاده از () می توان عبارت های منطقی متنوعی ساخت.

زبان برنامه نویسی فرضی ما دارای دستورات زیر است :

۱- تعریف متغیر

در زبان طراحی شده، متغیر ها دارای مقادیری هستند که در زمان compile مشخص خواهند شد. در این زبان datatype های موجود int, bool, var , float می باشند که مقداردهی اولیه ندارند. برای تعریف این متغیر ها از سینتکس های زیر استفاده می کنیم :

```
int x ;
bool y;
float a;
var c;
```

تایپ var به صورتی است که با استفاده از آن می توانید هر نوع متغیری تعریف کنید و بعد با توجه به مقداردهی شما کامپایلر تایپ آن متغیر را مشخص می کند.

```
var  x = 10 ;
var y = true ;
```

همچنین می توان چند متغیر از یک نوع تایپ را با هم به شکل زیر تعریف کرد :

```
int a,b,c = 1,2,3 ;
bool x,y,z = true, false , true ;
var  i , j , k = 10.01 , true ;
```

با توجه به مثال اخر متغیر i از نوع float و با مقدار اولیه ی 10.01 و متغیر j از نوع bool و مقدار اولیه true و متغیر k تایپ و مقدارش فعلا مشخص نیست.

برای نام گذاری متغیر ها تنها مجاز به استفاده از اعداد، حروف و "_" هستیم. همچنین نام متغیر نمی تواند با عدد شروع شود. دقت شود که کلیدواژه های موجود در زبان نمی تواند به عنوان نام یک متغیر به کار روند.
مثال هایی از نام گذاری های **اشتباه** :

```
int for ;
bool 12a ;
float _12a ;
var _a33 ;
```

- **تعریف متغیرهای ثابت**

```
const int MAX_VALUE = 100 ;
```

- **تعریف متغیر با define**

```
#define PI 3.14
int area  = PI * r * r ;
```

** در این پیاده سازی در نظر داشته باشید که خط اول عبارت (قسمت define) باید در ابتدای کد ها تعریف شود.

۲- انتساب متغیر

عملگرهای انتساب =- ، =+ ، =* ، =/ ، =% و = می توانند برای مقداردهی متغیر های int, float و var به کار روند اما برای متغیر های bool تنها مجاز به استفاده از = هستیم.

- **مقادیر مجاز متغیر bool**

حاصل عبارت های منطقی معتبر (استفاده از and, or , xor) و true , false

```
Is_null =  (false or ( 5 > = x ) ) xor 1;
```

- **مقادیر مجاز متغیر int , float**

اعداد و حاصل عبارت های ریاضی معتبر. همچنین اعداد می توانند با علامت مثبت، منفی و یا بدون علامت باشند. اما متغیر ها را نمی توانیم به صورت مستقیم با علامت - قرینه کنیم. برای قرینه کردن متغیر ها و عبارت ها باید پیش از پرانتز علامت - را قرار داد. برای مثال :

```
X = +1 - 2 * -( z^5 ) / - ( a + 33 ) ;
```

- **پیاده سازی unary operator**

عملگرهای یونری بر روی متغیرهای از نوع int و float و var های عددی قابل انجام باشند.

```
Y- -;
Y++;
```

- **پیاده سازی ternary operator**

برای مقدار دهی و انتساب همچنین باید فرمتی به شکل زیر توسط زبان قابل پذیرش باشد :

```
num = logic ? 1 : 2 ;
```

- **قابلیت cast کردن**

در انتساب نیز می توانید با کست کردن دیتا تایپ های مختلف به آن ها مقادیری بدهید. برای مثال :

```
float a = 1.23;
int  b = int ( a ) ;
int c = 1;
bool d = bool ( c ) ;
```

۳- پیاده سازی توابع ریاضی

توابع زیر را طوری پیاده سازی کنید تا در مثال های مختلف قابل استفاده باشند :

- **توابع min , max**

با استفاده از تابع max می توانیم بزرگترین عدد را برگردانیم و با استفاده از تابع min کوچکترین عدد را برمی گردانیم. ورودی این توابع، دو عدد است. خروجی از نوع float است.

- **تابع mean**

با استفاده از این تابع می توانیم میانگین اعداد یک بازه را برگردانیم. این تابع دو متغیر اول و اخر بازه را به عنوان ورودی گرفته و خروجی آن یک عدد به عنوان میانگین با تایپ float خواهد بود.

- **تابع sqrtN**

این تابع ریشه n ام یک عدد را محاسبه می کند و به صورت int بر می گرداند. (در ورودی تابع عدد و n داده می شود.)

۴- کامنت

در میان دستورات امکان کامنت گذاشتن وجود دارد. قابلیت کامنت تک‌خطی و چندخطی را داریم:

```
// single line comment
/* first comment
Second comment */
```

۵- دستورات شرطی

در داخل بلوک‌ها مثل حلقه‌ها و شرط‌ها امکان تعریف متغیر جدید وجود ندارد (چون در آن صورت نیاز به local scope داریم).

امکان ایجاد حلقه و شرط‌های تو در تو وجود داشته باشد.

- **if**

بلوک شرط با کلید واژه if آغاز می شود. سپس یک عبارت منطقی درون پرانتز نوشته شده و در ادامه آن، بدنه شرط در میان {} قرار می گیرد. در صورتی که حاصل عبارت منطقی true باشد، بدنه شرط اجرا می شود و در غیر

این صورت بدنه اجرا نمی شود. می توان به جای عبارت منطقی از یک متغیر bool و یا نوع دیگری از شرط که در ادامه توضیح می دهیم استفاده کرد.

پس از if می توانیم **چندین** else if و یا یک else نیز داشته باشیم :

```
if  ( x > 10 ) {

        A = 10;
}
else if ( y > 22 or z % 2 == 0 )
{
        A = 2 * y ;
}
else {
        A = A  - - ;
}
```

نوع دیگری از عبارت منطقی نیز می تواند به این شکل به عنوان جمله‌ی شرطی استفاده شود :

```
if (x in [ 1, 2 , 3 ])
```

این شرط زمانی اجرا می شود که x == 1 or x== 2 or x==3 باشد.

```
if  (x not in [ 1 , 2 , 3 ])
```

این شرط زمانی اجرا می شود که x!=1 and x!=2 and x!=3 باشد.

- **Switch case**

در این نوع از شرط ها نیز می خواهیم ساختاری درست مثل switch case که در c داریم پیاده سازی کنیم :

```
int var = 1 ;

switch ( var )
{
case 1 :
        var  = var + 1 ;
        break ;
case 2 :
        var  = var + 2 ;
        break ;
case  3 :
        var  = var + 3 ;
        break ;
default  :
        var  = var + 30 ;
        break;
}
```

دستورات break و continue در شرط و حلقه باید قابلیت پیاده سازی داشته باشد.

۶- دستورات حلقه

در داخل بلوک‌ها مثل حلقه‌ها و شرط‌ها امکان تعریف متغیر جدید وجود ندارد(چون در آن صورت نیاز به local scope داریم).

امکان ایجاد حلقه و شرط‌های تو در تو وجود داشته باشد.

- **do while**

```
do{
        X- -;
        print(Y+3);
}while( X >= 10 and Y%2 == 0)
```

دستورات break و continue در شرط و حلقه باید قابلیت پیاده سازی داشته باشد.

- **for**

سینتکس این حلقه به شکل زیر است :

```
int i;
for  ( i = 0 ; i < n ; i ++ ) {
        if(i < 3){
                continue;
        }
        print(d);
 }
```

دستورات break و continue در شرط و حلقه باید قابلیت پیاده سازی داشته باشد.

۷- تابع print

پرینت کردن پاسخ عبارت‌های ریاضی، منطقی و متغیرها.

```
print(1+2)
// 3 should be printed
print(A)
// value of A should be printed
```

کدی که توسط کامپایلر شما کامپایل می شود علاوه بر syntax صحیح، باید semantic درستی نیز داشته باشد. بنابراین به نکات زیر توجه داشته باشید :

- قبل از آن که متغیری مقدار بگیرد، باید تعریف شده باشد.
- تعریف متغیری که قبلا تعریف شده است مجاز نیست،
- نمی توان به متغیر های ثابت و متغیرهایی که با define تعریف شده باشند، مقدار جدیدی را انتساب کرد.
- متغیر های var فقط در صورتی که از یک نوع باشند می توانند در یک عملیات ریاضی یا منطقی قرار بگیرند.
- متغیر هایی از یک نوع را نمی توان داخل متغیرهایی از نوع دیگر ریخت مگر از casting استفاده شده باشد.

نکات مهم

- از زبان C یا C++ برای توسعه کدها استفاده کنید.
- توسعه هر بخش در فایل‌های جداگانه انجام شود که قابلیت ارزیابی مجزا را داشته باشد.
- کدها خوانایی مناسبی داشته باشند و پیشنهاد می‌شود به درستی کامنت‌گذاری شود.
- پروژه به صورت تیمی قابل انجام است. اندازه تیم‌ها بین ۲ الی ۳ نفر قابل قبول است و در فاز بعدی، امکان تغییر اعضای تیم وجود ندارد.
- همه اعضای تیم باید در انجام پروژه مشارکت داشته باشند و تسلط هر فرد جداگانه ارزیابی خواهد شد.
- جهت پیاده‌سازی درست و کامل پروژه، پیشنهاد می‌شود اسناد مرتبط با سایت LLVM با دقت مطالعه شده و همچنین سه فصل اول کتاب learn llvm 12 مطالعه شود. کدهای نمونه و روش ارائه مطالب در کتاب به درک شما از پیاده‌سازی کامپایلر مینی‌مال کمک خواهدکرد.
- بخش قابل توجهی از نمره پروژه، مربوط به اجرای صحیح تست‌ها است که در زمان ارائه به شما داده خواهد شد. صحت عملکرد هر بخش از پروژه نیز جداگانه بررسی می‌شود و در صورتی که در بخشی از پروژه اشکال داشته باشید، نمره‌ی باقی قسمت‌ها را دریافت می‌کنید.

موفق باشید